**Capítulo 18  
Privacidade em Blockchain**

Neste capítulo, abordaremos a privacidade, que é um tópico importante. A privacidade de dados ou privacidade da informação não é um conceito novo. A privacidade de dados refere-se à proteção de informações pessoais e sensíveis e aos direitos dos indivíduos de controlar como suas informações pessoais são coletadas, armazenadas, usadas e compartilhadas. Embora a privacidade sempre tenha sido importante, com o advento da internet e dos serviços online, proteger as informações pessoais tornou-se ainda mais crucial do que nunca. No mundo digital estabelecido, como a internet e outros serviços online, existem muitas leis e regulamentações, juntamente com políticas rigorosas de segurança da informação e controles técnicos que protegem os usuários.

No entanto, no mundo blockchain, há uma clara falta de tal rigor até o momento. Alcançar o mesmo nível de segurança que o mundo digital típico desfruta ainda é uma questão que precisa ser abordada. Neste capítulo, veremos o que é privacidade no contexto do blockchain e como podemos introduzi-la usando diferentes protocolos. Ao longo do caminho, examinaremos os seguintes tópicos:

* Privacidade e seus tipos
* Protocolos de Camada 0, Camada 1 e Camada 2 para privacidade em blockchain
* Provas de conhecimento zero, seus vários tipos, esquemas de compromisso polinomial e protocolos relevantes
* Um exemplo prático

Então, vamos começar; primeiro discutiremos a privacidade, por que ela é importante e seus vários tipos.

**Privacidade**

A privacidade no blockchain pode ser dividida em duas categorias principais com base no tipo de serviço requerido. Essas categorias são o anonimato dos usuários e a confidencialidade das transações. O anonimato está relacionado ao ocultamento da identidade do remetente ou do destinatário, enquanto a confidencialidade aborda os requisitos de ocultar os valores das transações.

A razão fundamental pela qual os blockchains não preservam a privacidade é que cada transação em um blockchain precisa ser verificada e executada por cada participante na rede. Embora essa propriedade de execução de todas as transações por todos os nós conceda aos blockchains sua poderosa propriedade de integridade e garantia de execução, isso também é uma fraqueza.

Essa fraqueza é resultado da transparência inerente nos blockchains. Como todos os dados das transações, incluindo detalhes da conta, entradas, saídas e estados, são visíveis para qualquer pessoa no blockchain, a privacidade não pode ser preservada.

Uma solução que vem à mente é que poderíamos, de alguma forma, criptografar os dados, mas se os valores estiverem ocultos, então as transações não poderão ser verificadas. O requisito fundamental é, de algum modo, combinar verificabilidade pública e confidencialidade, de modo que, mesmo que os dados estejam criptografados de alguma forma, ainda assim possam ser verificados publicamente.

Vamos agora explicar anonimato e confidencialidade, antes de aprofundarmos em o que podemos fazer para resolver esse problema.

**Anonimato**

O anonimato é desejável em situações em que a identidade dos usuários precisa ser ocultada dos outros participantes em uma rede. Isso pode se dever a exigências regulatórias, exigências empresariais, ou simplesmente à natureza sensível das transações. Por exemplo, em uma rede empresarial, pode ser desejável ocultar as negociações entre diferentes participantes da rede de outros concorrentes, para evitar fricções ou vantagem competitiva injustificada.

As propriedades de não vinculabilidade (*unlinkability*) e ser não rastreável (*untraceable*) são usadas para alcançar o anonimato. A propriedade de não rastreabilidade nos permite ocultar o rastro de uma transação de uma parte para outra em uma rede. Por outro lado, a não vinculabilidade significa que um observador é incapaz de deduzir a ligação entre transações e seus participantes, ou os relacionamentos entre os participantes das transações (remetentes e destinatários). Isso significa que, se uma terceira parte observar o fluxo de transações, ela não conseguirá ver quais entidades estão envolvidas em transações com quais outras entidades.

Existem vários blockchains que suportam privacidade de forma inerente e que são construídos com essa funcionalidade no projeto do sistema. O mais popular é o Zcash, que usa provas de conhecimento zero (*zero-knowledge proofs – ZKPs*) para alcançar o anonimato. Outros exemplos incluem Monero (<https://web.getmonero.org>), que utiliza assinaturas em anel (*ring signatures*) para fornecer serviços de anonimato.

**Confidencialidade**

A confidencialidade é um requisito absoluto em muitas indústrias, como finanças, direito e saúde. Da mesma forma, a privacidade das transações é uma propriedade altamente desejada nos blockchains. No entanto, devido à sua própria natureza, especialmente nos blockchains públicos, tudo é transparente, o que inibe seu uso em várias indústrias onde a privacidade é de importância primordial, como finanças, saúde e muitas outras.

Quando pensamos em confidencialidade, podemos dividi-la em dois requisitos opcionais adicionais: privacidade condicional e divulgação seletiva. Privacidade condicional significa que um sistema deve ter a capacidade de tornar os dados visíveis condicionalmente para uma terceira parte, como auditores, mas manter os dados ocultos de todas as outras partes, exceto daquelas que têm conhecimento das transações. Note que a privacidade incondicional também não é boa na prática, porque, se tudo estiver oculto de todos, isso pode permitir atividades criminosas como lavagem de dinheiro e financiamento ao terrorismo, sem que ninguém saiba disso. Portanto, é necessário, especialmente nas aplicações financeiras de blockchain, garantir que, em alguns casos, uma terceira parte como um regulador ou auditor possa acessar e visualizar os dados para assegurar a conformidade com os requisitos regulatórios.

O outro tipo de confidencialidade é a **divulgação seletiva**, onde um sistema deve ter a capacidade de compartilhar seletivamente alguma parte dos dados, como apenas a idade, a partir de um conjunto de dados maior que contém informações pessoais. Uma variação disso é chamada de **prova de intervalo** (*range proof*), que é a capacidade de provar que alguns dados estão dentro de um intervalo, por exemplo, minha capacidade de provar que tenho mais de 18 anos de idade ou que meu salário está entre $10.000 e $20.000.

Agora veremos algumas técnicas que podemos usar para alcançar privacidade.

**Técnicas para alcançar privacidade**

Como o blockchain é um livro-razão público de todas as transações e está abertamente disponível, torna-se fácil analisá-lo. Quando combinado com análises de tráfego, as transações podem ser vinculadas de volta aos seus endereços IP de origem, revelando possivelmente o originador de uma transação. Esta é uma grande preocupação do ponto de vista da privacidade.

No domínio do Bitcoin, é uma prática recomendada e comum gerar um novo endereço para cada transação, o que permite algum nível de não vinculabilidade. No entanto, isso não é suficiente, e várias técnicas foram desenvolvidas e usadas com sucesso para rastrear o fluxo de transações através da rede e vinculá-las de volta ao seu originador. Essas técnicas analisam blockchains usando grafos de transação, grafos de endereços e grafos de entidades, os quais facilitam a vinculação de usuários às transações, levantando assim preocupações de privacidade.

Essas técnicas baseiam-se na ideia de que cada transação na rede Bitcoin (ou em qualquer blockchain público) é registrada publicamente e pode ser vinculada a outras transações, endereços e entidades. Por exemplo, a análise de grafo de transações envolve a criação de um grafo de todas as transações e a identificação de padrões de transações entre endereços. A análise de grafo de endereços envolve a criação de um grafo de todos os endereços e suas transações associadas. Esse tipo de análise pode ajudar a identificar os relacionamentos entre diferentes endereços e o fluxo de fundos entre eles.

As técnicas mencionadas anteriormente podem ser ainda enriquecidas usando informações publicamente disponíveis (por exemplo, endereços Bitcoin de usuários em fóruns públicos da internet) sobre transações e vinculando-as aos usuários reais. Existem analisadores de blocos (*block parsers*) de código aberto disponíveis que podem ser usados para extrair informações de transações, saldos e scripts do banco de dados da blockchain.

Nesta seção, descreveremos diferentes técnicas para alcançar privacidade, incluindo anonimato e confidencialidade.

Assim como soluções de escalabilidade, também podemos dividir as soluções de privacidade em três categorias, com base na camada dentro da pilha de arquitetura do blockchain na qual operam:

* **Métodos de Camada 0**, ou métodos de camada de rede, onde o mecanismo para alcançar privacidade opera no nível da rede.
* **Métodos de Camada 1**, também chamados de métodos *on-chain*, onde o próprio protocolo do blockchain é aprimorado para alcançar privacidade.
* **Métodos de Camada 2**, ou métodos *off-chain*, onde mecanismos que existem fora do blockchain principal são usados para alcançar privacidade no blockchain.

**Camada 0**

Essas soluções são métodos no nível da camada de rede que fornecem anonimato utilizando **TOR** e **I2P**. Esses métodos nos permitem ocultar as identidades das partes envolvidas.

**Tor**

Tor, *The Onion Router*, é um software que permite comunicações anônimas. Mais informações sobre o Tor estão disponíveis aqui: <https://www.torproject.org>. Podemos usar o Tor para permitir comunicação anônima em redes blockchain de criptomoedas. Tor é uma escolha comum para permitir comunicação anônima e tem sido usado em Monero, Verge e no Bitcoin para fornecer anonimato.

Para Bitcoin, veja: <https://en.bitcoin.it/wiki/Tor>. O Monero também pode ser executado com o Tor (<https://web.getmonero.org/>). Verge é outro exemplo de criptomoeda que faz uso do Tor para ofuscação de IP.

**I2P**

**I2P**, *Invisible Internet Project*, é uma rede anônima construída sobre a internet. Ela permite comunicação peer-to-peer resistente à censura. É usada no Monero para fornecer serviços de anonimato. Mais informações sobre o I2P estão disponíveis aqui: <https://geti2p.net/en/>. Monero e Zcash suportam transações anônimas. O Monero faz uso de assinaturas em anel, endereços furtivos e transações confidenciais, enquanto o Zcash usa zk-SNARKs.

Como muitas das técnicas descritas nas seções seguintes podem ser usadas nas Camadas 1 e 2 — por exemplo, ZKPs — não estamos distinguindo entre essas camadas. Em vez disso, listaremos apenas essas soluções e discutiremos para que elas podem ser usadas. Além disso, as soluções para anonimato e confidencialidade também se sobrepõem, e as técnicas usadas para alcançar confidencialidade também são mais ou menos aplicáveis para alcançar anonimato. No entanto, separamos a Camada 0, a camada de rede, e descreveremos algumas das abordagens no nível de rede para alcançar anonimato, antes de apresentar uma variedade de mecanismos de privacidade aplicáveis às Camadas 1 e 2.

**Ofuscação por Indistinguibilidade**

Essa técnica criptográfica pode servir como uma solução milagrosa para todos os problemas de privacidade e confidencialidade nos blockchains, mas a tecnologia ainda não está pronta para implementações em produção. **Ofuscação por indistinguibilidade** (*Indistinguishability Obfuscation – IO*) permite a ofuscação de código, que é um tópico de pesquisa muito ativo em criptografia. Se aplicada aos blockchains, a IO pode servir como um mecanismo de ofuscação inquebrável que transforma contratos inteligentes em uma caixa-preta onde o comportamento do código ofuscado é indistinguível. Em termos mais simples, a funcionalidade interna do contrato inteligente é totalmente oculta.

A ideia chave por trás da IO é o que os pesquisadores chamam de “*quebra-cabeça multilinear*”, que basicamente ofusca o código do programa misturando-o com elementos aleatórios, e se o programa for executado conforme pretendido, produzirá a saída esperada. No entanto, qualquer outra maneira de executá-lo fará com que o programa produza dados aleatórios e sem sentido.

Em outras palavras, a IO torna computacionalmente inviável para um atacante distinguir entre duas execuções diferentes do programa, mesmo que o atacante tenha acesso completo ao código do programa e ao comportamento de entrada/saída.

Em outras palavras, a IO fornece uma maneira de proteger a privacidade da implementação de um programa, tornando impossível para um atacante determinar os detalhes exatos de como o programa é executado ou quais entradas ele recebeu.

Este artigo de pesquisa está disponível em: <https://eprint.iacr.org/2013/451.pdf>.

**Criptografia homomórfica**

Este tipo de criptografia permite que operações sejam realizadas sobre dados criptografados. Imagine um cenário em que os dados são enviados para um servidor em nuvem para processamento. O servidor os processa e retorna a saída sem saber nada sobre os dados que foram processados. Esta também é uma área madura para pesquisa, e a **criptografia homomórfica totalmente funcional**, que permite todas as operações sobre dados criptografados, ainda não é totalmente implantável em produção; no entanto, já houve progressos significativos neste campo. Uma vez implementada em blockchains, ela pode permitir o processamento de transações criptografadas sem a necessidade de descriptografá-las, o que permitirá, por si só, a privacidade e a confidencialidade das transações.

Por exemplo, os dados armazenados no blockchain podem ser criptografados usando criptografia homomórfica, e cálculos podem ser realizados sobre esses dados sem a necessidade de descriptografá-los, proporcionando assim um serviço de privacidade sobre blockchains. Este conceito também foi implementado em um projeto chamado **Enigma**, que está disponível online em: <https://www.media.mit.edu/projects/enigma/overview/>, pelo Media Lab do MIT. O Enigma é uma rede ponto a ponto que permite a várias partes realizar cálculos sobre dados criptografados sem revelar nada sobre os dados. A pesquisa está disponível em: <https://crypto.stanford.edu/craig/>.

**Computação multipartidária segura**

O conceito de **computação multipartidária segura** (*secure multiparty computation*) não é novo e é baseado na noção de que os dados são divididos em múltiplas partições entre as partes participantes, sob um mecanismo de compartilhamento secreto, que então realiza o processamento real sobre os dados sem a necessidade de reconstruir os dados em uma única máquina.

A saída produzida após o processamento também é compartilhada entre as partes. Várias partes realizam o cálculo mutuamente sem revelar suas entradas. Somente a saída da computação é revelada.

Uma plataforma de computação multipartidária segura chamada **Enigma** foi proposta em 2015. O artigo está disponível aqui: <https://web.media.mit.edu/~guyzys/data/enigma_full.pdf>.

**Confidencialidade assistida por hardware confiável**

Plataformas de computação confiável podem ser usadas para fornecer um mecanismo pelo qual a confidencialidade de uma transação pode ser alcançada em um blockchain, por exemplo, usando **Intel Software Guard Extensions (SGX)**, que permite que código seja executado em um ambiente protegido por hardware chamado *enclave*. Uma vez que o código seja executado com sucesso no enclave isolado, ele pode produzir uma prova, chamada *quote*, que pode ser atestada pelos servidores em nuvem da Intel.

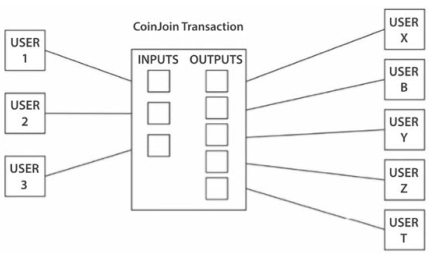
Há uma preocupação de que confiar na Intel resulte em algum nível de centralização e não esteja em linha com o verdadeiro espírito da tecnologia blockchain. No entanto, essa solução tem seus méritos e, na realidade, muitas plataformas já usam chips da Intel, então confiar na Intel pode ser aceitável por alguns em certos casos.

Se essa tecnologia for aplicada a contratos inteligentes, então, uma vez que um nó tenha executado o contrato inteligente, ele pode produzir uma prova de correção, demonstrando a execução bem-sucedida, e os outros nós apenas precisarão verificá-la. Essa ideia pode ser estendida ainda mais usando qualquer **Ambiente de Execução Confiável (TEE – Trusted Execution Environment)**, que pode fornecer a mesma funcionalidade de um enclave e está disponível até mesmo em dispositivos móveis com Comunicação por Campo de Proximidade (NFC – Near Field Communication) e um elemento seguro. Por exemplo, **Ekiden** é uma plataforma que faz uso do TEE do Intel SGX para executar contratos inteligentes enquanto preserva a confidencialidade. Mais informações sobre o Ekiden estão disponíveis aqui: <https://arxiv.org/pdf/1804.05141>.

**Protocolos de mistura**

Um **protocolo de mistura**, ou *mixer*, é um serviço que permite aos usuários preservar sua privacidade misturando suas moedas com as de outros usuários.

Esses esquemas são usados para fornecer anonimato às transações de criptomoedas. Nesse modelo, é usado um provedor de serviço de mistura (um intermediário ou uma carteira compartilhada). Os usuários enviam moedas para essa carteira compartilhada como um depósito, e então a carteira compartilhada pode enviar outras moedas (do mesmo valor depositado por outros usuários) ao destino. Os usuários também podem receber moedas que foram enviadas por outros por meio desse intermediário. Dessa forma, o vínculo entre saídas e entradas não existe mais, e a análise do grafo de transações não será capaz de revelar a relação real entre remetentes e destinatários:



**Coinjoin** é um exemplo de protocolo de mistura, onde duas transações são unidas para formar uma única transação, mantendo as entradas e saídas inalteradas. A ideia central por trás do Coinjoin é construir uma transação compartilhada que seja assinada por todos os participantes. Essa técnica melhora a privacidade de todos os participantes envolvidos nas transações.

Coinjoin é uma técnica usada para anonimizar transações em Bitcoin misturando-as de forma interativa. A ideia baseia-se na formação de uma única transação a partir de múltiplas entidades, sem causar nenhuma alteração nas entradas e saídas. Ela remove o vínculo direto entre remetentes e destinatários, o que significa que um único endereço não pode mais ser associado a transações, o que poderia levar à identificação de usuários.

O Coinjoin requer cooperação entre várias partes dispostas a criar uma única transação misturando pagamentos. Portanto, deve-se observar que, se qualquer participante no esquema Coinjoin não cumprir o compromisso feito de cooperar na criação de uma transação única — ao não assinar as transações conforme exigido — isso pode resultar em um ataque de negação de serviço (DoS).

Nesse protocolo, não há necessidade de uma terceira parte confiável única. Este conceito é diferente de um serviço de mistura, que atua como uma terceira parte confiável ou intermediária entre os usuários de Bitcoin e permite o embaralhamento das transações. Esse embaralhamento de transações resulta na prevenção de rastreamento e vinculação dos pagamentos a um usuário específico.

Vários serviços de mistura de terceiros também estão disponíveis, mas, se um serviço for centralizado, ele representa a ameaça de rastreamento do mapeamento entre remetentes e destinatários. Isso porque o serviço de mistura conhece todas as entradas e saídas. Além disso, misturadores totalmente centralizados apresentam o risco de que os administradores do serviço roubem as moedas.

Diversos serviços, com graus variados de complexidade, como **CoinShuffle**, **Coinmux** e **Darksend** no Dashcoin, estão disponíveis, baseando-se na ideia de transações Coinjoin (mistura). O **CoinShuffle** é uma alternativa descentralizada aos serviços de mistura tradicionais, pois não requer uma terceira parte confiável.

No entanto, os esquemas baseados em Coinjoin possuem algumas fraquezas, mais notavelmente a possibilidade de lançar um ataque DoS por usuários que se comprometeram a assinar as transações inicialmente, mas agora não fornecem sua assinatura, atrasando ou interrompendo completamente as transações conjuntas.

**CoinSwap**

**CoinSwap** é um mecanismo de privacidade baseado na ideia de trocas atômicas (*atomic swaps*). As trocas atômicas permitem que duas partes troquem moedas sem exigir uma terceira parte confiável. O CoinSwap também pode ser usado para trocas entre blockchains diferentes (*cross-chain swaps*). O CoinSwap funciona utilizando uma terceira parte no fluxo da transação e também requer canais de comunicação privados entre todas as partes. Dessa forma, os endereços do remetente e do destinatário não podem ser vinculados. Essa terceira parte recebe fundos do remetente e os envia ao destinatário.

A técnica aqui é que a terceira parte paga os fundos ao destinatário usando uma fonte de fundos totalmente diferente, desconectando assim o vínculo entre o remetente e o destinatário. Essa desconexão resulta em uma transação não vinculável e, portanto, proporciona privacidade. O remetente usa transações multiassinadas (normalmente 2 de 2 — remetente e terceira parte) para permitir que as transações sejam gastas, enquanto o destinatário também requer transações multiassinadas (normalmente 2 de 2 — destinatário e terceira parte) para que as transações sejam gastas. Em outras palavras, o remetente e a terceira parte assinarão a saída da transação para enviar o Bitcoin à terceira parte, enquanto o destinatário e a terceira parte assinarão a saída da transação para enviar o Bitcoin ao destinatário.

O CoinSwap usa transações bloqueadas por hash (*hash-locked transactions*) em que uma pré-imagem do hash é necessária para desbloquear a transação. O uso de transações bloqueadas por hash impede que a terceira parte roube o Bitcoin. Mais informações sobre o CoinSwap podem ser encontradas no link a seguir, onde ele foi originalmente proposto por Gregory Maxwell: <https://bitcointalk.org/index.php?topic=321228>.

**TumbleBit**

O protocolo **TumbleBit** foi introduzido em 2016. O TumbleBit é totalmente compatível com o protocolo do Bitcoin. Trata-se de um protocolo de pagamentos fora da cadeia (*off-chain*), anônimo, rápido e com *unlinkability*, que permite que as partes transfiram fundos por meio de uma terceira parte ou intermediário não confiável, chamado de *tumbler*. Neste protocolo, até mesmo o *tumbler* é incapaz de desanonimizar os pagadores e recebedores envolvidos em um pagamento. Ele envolve o uso de dois protocolos de troca justa que impedem qualquer atividade maliciosa, como trapaça por participantes ou pelo *tumbler*. O TumbleBit se baseia em um protocolo chamado **solucionador de quebra-cabeça RSA** (*RSA puzzle solver*), que permite a um pagador fazer pagamentos ao *tumbler*. A menos que o *tumbler* resolva esse quebra-cabeça RSA, ele não pode reivindicar nenhum Bitcoin pago pelo pagador. Outro protocolo de troca justa, chamado **protocolo de promessa de quebra-cabeça** (*puzzle-promise protocol*), é usado entre o *tumbler* e o recebedor para reivindicar o pagamento.

O TumbleBit consiste em três fases, conforme listado aqui:

1. **Fase de depósito em garantia (escrow)**, onde todos os canais de pagamento são configurados.
2. **Fase de pagamentos**, onde os pagadores transferem fundos.
3. **Fase de retirada (cash-out)**, onde pagadores e recebedores encerram os canais de pagamento.

Mais informações sobre o TumbleBit podem ser encontradas aqui: <https://eprint.iacr.org/2016/575.pdf>. Uma implementação de prova de conceito do TumbleBit está disponível aqui: <https://github.com/BUSEC/TumbleBit>.

Uma inovação recente é a introdução do **TumbleBit++**, que é um protocolo de mistura aprimorado baseado no TumbleBit. Ele fornece anonimato e confidencialidade dos valores das transações combinando transações confidenciais e um *hub* de pagamento anônimo centralizado e não confiável. Mais informações sobre o TumbleBit++ estão disponíveis aqui: <https://link.springer.com/chapter/10.1007/978-3-030-31919-9_21>.

Agora discutiremos o **Dandelion**, que fornece anonimato inerente e é implementado por meio da reformulação da camada de rede do protocolo do Bitcoin.

**Dandelion**

Além das abordagens mencionadas anteriormente, uma proposta recente chamada **Dandelion** também foi apresentada. O Dandelion (sua versão aprimorada é chamada **Dandelion++**) é uma proposta que visa tornar as transações na rede do Bitcoin intraceáveis. Este protocolo permitirá que transações anônimas ocorram na rede Bitcoin, em oposição a transações pseudônimas, onde um adversário, usando métodos de análise de rede, pode rastrear a transação até seu nó de origem. Consequentemente, o adversário pode então descobrir o endereço IP original do remetente da transação.

Podemos dizer que o Dandelion é um mecanismo para fornecer anonimato inerente às transações em Bitcoin porque, uma vez implementado, a camada P2P do protocolo de rede do Bitcoin será modificada de tal forma que rastrear transações até seu nó de origem e IP se tornará extremamente difícil.

A desanonimização de usuários de Bitcoin é um problema conhecido, e diversos artigos de pesquisa estão disponíveis sobre esse tema. Por exemplo, o artigo *Deanonymization of clients in Bitcoin P2P network* está disponível em: <https://arxiv.org/pdf/1405.7418.pdf>.

Uma proposta de melhoria do Bitcoin está disponível em:  
<https://github.com/bitcoin/bips/blob/master/bip-0156.mediawiki>.  
O artigo original da proposta do Dandelion está disponível em:  
<https://arxiv.org/pdf/1701.04439.pdf>.  
O artigo de pesquisa sobre o Dandelion++ está disponível em:  
<https://arxiv.org/pdf/1805.11060.pdf>.

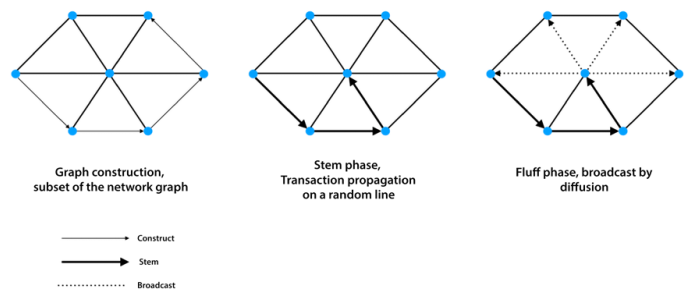
Na rede Bitcoin, o mecanismo de propagação epidêmica (*epidemic flooding mechanism*, também conhecido como **protocolo de fofoca – Gossip protocol**) é utilizado para propagação de transações. Sobre esse mecanismo, há um método relativamente eficaz chamado **difusão**, que é utilizado para prover algum nível de anonimato. Nesse protocolo, cada nó introduz atrasos independentes e exponenciais para espalhar transações aos seus vizinhos. Esse esquema resulta na redução da simetria do protocolo epidêmico, o que torna a análise da rede mais difícil e menos confiável. No entanto, esse esquema é previsível e, portanto, não fornece garantias suficientes de anonimato.

O Dandelion propõe um novo mecanismo de roteamento compatível com versões anteriores. Nesse protocolo:

* Primeiro, é construído um grafo de privacidade (*privacy graph*, também chamado de **conjunto de anonimato**). Esta fase é chamada de **fase de construção do grafo privado**. Trata-se de um subgrafo da rede P2P existente do Bitcoin, e cada nó seleciona um subconjunto de seus pares de saída nesta fase. É nesse momento que a linha aleatória de nós é selecionada.
* As mensagens (transações) são então roteadas por esse grafo de privacidade durante a **fase do Caule – Stem phase**. É aqui que a mensagem é propagada por uma linha aleatória de nós.
* Finalmente, há a **fase da Flor – Fluff phase**, onde as mensagens são roteadas (transmitidas) para toda a rede por difusão.

Em resumo, o protocolo Dandelion é composto por uma fase de anonimato e uma fase de propagação. Na fase de anonimato, esse protocolo espalha uma mensagem ao longo de uma linha aleatória por um número aleatório de saltos. Após esse passo, a mensagem é transmitida para toda a rede usando difusão. Com essa combinação de seleção de caminho aleatório e difusão, o protocolo Dandelion fornece garantias de anonimato quase ideais.

Esse protocolo pode ser visualizado utilizando o seguinte diagrama:



*Figure 18.2: Dandelion protocol routing*

**Transações confidenciais**

As transações confidenciais fazem uso de **compromissos de Pedersen** (*Pedersen commitments*) para fornecer confidencialidade. Esquemas de compromisso permitem que um usuário se comprometa com um determinado valor enquanto o mantém em segredo, com a capacidade de revelá-lo posteriormente. Esquemas de compromisso podem ser construídos simplesmente usando funções de hash criptográficas. Duas propriedades precisam ser satisfeitas para projetar um esquema de compromisso: **vinculação** (*binding*) e **ocultação** (*hiding*).

A vinculação garante que o compromissário não consiga alterar o valor escolhido depois de comprometido, enquanto a propriedade de ocultação garante que qualquer adversário não seja capaz de descobrir o valor original ao qual o compromissário se comprometeu.

Um compromisso de Pedersen é um tipo de esquema de compromisso informacionalmente oculto e computacionalmente vinculante, sob a suposição do logaritmo discreto. Compromissos de Pedersen permitem operações adicionais e preservam a propriedade comutativa nos compromissos, o que os torna particularmente úteis para fornecer confidencialidade em transações de Bitcoin. Em outras palavras, eles suportam criptografia homomórfica parcial de valores, o que significa que, usando esquemas de compromisso, podemos ocultar valores de pagamento em uma transação de Bitcoin. Esse conceito já foi implementado no Projeto Elements:  
<https://elementsproject.org/features/confidential-transactions>.

**MimbleWimble**

O esquema **MimbleWimble** foi proposto anonimamente no canal IRC do Bitcoin e desde então ganhou muita popularidade. O MimbleWimble estende a ideia de transações confidenciais e Coinjoin, o que permite a agregação de transações sem exigir nenhuma interatividade. No entanto, ele não suporta o uso da linguagem de scripts do Bitcoin, juntamente com várias outras funcionalidades do protocolo padrão do Bitcoin. Isso o torna incompatível com o protocolo atual do Bitcoin. Portanto, ele pode ser implementado como uma **sidechain** do Bitcoin ou como uma criptomoeda alternativa. No entanto, ele foi implementado no **Litecoin** em maio de 2022.

Esse esquema pode abordar simultaneamente problemas de privacidade e escalabilidade. Os blocos criados usando a técnica MimbleWimble não contêm transações como nos blockchains tradicionais do Bitcoin; em vez disso, esses blocos são compostos por três listas: uma lista de entradas, uma lista de saídas e algo chamado de **excedentes** (*excesses*), que são listas de assinaturas e diferenças entre saídas e entradas. A lista de entradas basicamente faz referência às saídas antigas, e a lista de saídas contém saídas de transações confidenciais.

Os blocos criados usando o esquema MimbleWimble são verificáveis pelos nós utilizando assinaturas, entradas e saídas para assegurar a legitimidade do bloco. Em contraste com o Bitcoin, as saídas das transações no MimbleWimble contêm apenas chaves públicas, e a diferença entre as saídas antigas e novas é assinada por todos os participantes envolvidos nas transações.

**ZkLedger**

Este livro-razão distribuído fornece privacidade de transação, incluindo confidencialidade do valor da transação e dos endereços do remetente e do destinatário. Ele utiliza **provas de conhecimento zero não interativas (NIZKs)** e **protocolos Sigma** para alcançar isso. No entanto, este protocolo não é escalável porque o tamanho das transações cresce linearmente com o número de partes envolvidas.

**Criptografia baseada em atributos**

A **criptografia baseada em atributos (ABE – Attribute-Based Encryption)** é um tipo de criptografia de chave pública que fornece simultaneamente confidencialidade e controle de acesso. A ideia principal por trás deste esquema é que a chave privada de um usuário e o seu texto cifrado dependem dos atributos do usuário, como sua localização, fuso horário ou seu papel na organização. Isso significa que a descriptografia só é possível quando não apenas a chave privada está disponível, mas também os atributos correspondentes estão presentes.

**Assinaturas anônimas**

Assinaturas anônimas são tipos de assinaturas digitais nas quais as assinaturas não revelam a identidade do signatário. Existem principalmente dois esquemas disponíveis para assinaturas anônimas: **assinaturas de grupo** e **assinaturas em anel (ring signatures)**.

As **assinaturas de grupo** permitem que um conjunto de signatários forme um grupo gerenciado por um gerente de grupo. Cada membro do grupo recebe uma chave de assinatura de grupo fornecida pelo gerente. Essa chave de assinatura de grupo permite que cada membro do grupo assine mensagens anonimamente em nome do grupo. O gerente de grupo é capaz de identificar quem assinou a mensagem, enquanto entidades externas não conseguem. Embora as assinaturas de grupo possam funcionar bem, uma das limitações deste esquema é que o gerente do grupo pode identificar os usuários. Esse problema foi abordado com as assinaturas em anel.

As **assinaturas em anel** permitem que um conjunto de signatários forme um grupo (um anel) de membros. Cada membro deste grupo é capaz de assinar mensagens em nome do anel. Diferentemente das assinaturas de grupo, **não há gerente de grupo** nas assinaturas em anel, de modo que **ninguém é capaz de identificar os signatários**.

Embora todas essas técnicas sejam úteis e forneçam muitas propriedades desejáveis de anonimato, também há um problema com o uso malicioso dessa tecnologia. Imagine se o anonimato for utilizado por criminosos envolvidos em lavagem de dinheiro ou na venda de drogas ilegais usando criptomoedas anônimas. Seria quase impossível rastrear atividades ilegais até seu originador se o anonimato absoluto for alcançado. Imagine uma variante do mercado **Silk Road** (<https://en.wikipedia.org/wiki/Silk_Road_(marketplace)>) usando uma criptomoeda totalmente anônima e confidencial. Como isso poderia ser rastreado e interrompido?

Até agora discutimos técnicas criptográficas genéricas e alguns exemplos para permitir a privacidade no blockchain. No entanto, ainda não tocamos no conceito importante de **contratos inteligentes privados**. Blockchains programáveis, começando com o Ethereum e agora com cadeias como Solana e Polkadot, todos suportam contratos inteligentes. É desejável alcançar privacidade também no nível do contrato inteligente, em vez de apenas no nível da transação padrão. Pense em um contrato ERC-20 onde todas as operações, como transferências, verificação de saldo e outras, são realizadas com total privacidade. Existem várias propostas para isso também, algumas das quais discutiremos a seguir.

**Zether**

**Zether** permite um mecanismo de transação privada que oferece confidencialidade e anonimato. Ele é implementado na blockchain do Ethereum, mas, em teoria, pode ser implementado em qualquer blockchain programável. Zether é implementado como um contrato inteligente no qual o ether é depositado e transformado em um token Zether.

Outras propostas incluem:

* **Zexe** (<https://eprint.iacr.org/2018/962.pdf>)
* **PPChain** (<https://ieeexplore.ieee.org/document/9199417>)
* **Hawk** (<https://ieeexplore.ieee.org/document/7546538>)
* **ZKLedger** (<https://dci.mit.edu/zkledger>)

**Privacidade usando protocolos de Camada 2**

A privacidade usando Camada 2 e **canais de estado** (*state channels*) também é possível, simplesmente porque todas as transações são executadas fora da cadeia (*off-chain*), e a blockchain principal não vê as transações, exceto pelo estado final de saída e dados mínimos de transação (representando transações), o que garante anonimato e confidencialidade. No entanto, observe que, geralmente, a privacidade **não é o objetivo principal dos protocolos de Camada 2** — eles se concentram mais em fornecer escalabilidade. Mesmo assim, a Camada 2 pode ser usada para oferecer anonimato e confidencialidade se o conhecimento zero for usado não apenas como uma prova de computação correta (integridade), mas também como uma **medida de confidencialidade**. Isso é realizado no projeto **Aztec** (<https://aztec.network>).

**Gerenciadores de privacidade**

**Gerenciadores de privacidade** fornecem um mecanismo que garante a confidencialidade das transações. Estes são componentes fora da cadeia (*off-chain*) que substituem a carga útil da transação por um índice hash, de tal forma que **apenas os participantes da transação** conseguem encontrar a carga útil criptografada correspondente representada pelo índice hash. Outras partes que não participam da transação simplesmente ignoram o hash. Este conceito foi discutido no **Capítulo 16**, sobre **Blockchain Empresarial** (*Enterprise Blockchain*).

Além da escalabilidade e da privacidade, há vários outros aspectos gerais de segurança que precisam ser abordados no blockchain. Descreveremos esses tópicos de segurança geral no próximo capítulo, **Capítulo 19 – Segurança em Blockchain**.

Existem duas abordagens quando se trata de soluções de escalabilidade e privacidade:

1. A primeira é o desenvolvimento de **linguagens específicas de domínio (DSLs – Domain Specific Languages)**, que podem ser usadas para escrever programas que podem ser verificados por um verificador na Camada 1.
2. A segunda são as **máquinas virtuais de conhecimento zero (ZK VMs)**, que permitem que computações (programas) sejam verificadas na Camada 2.

DSLs exigem que os desenvolvedores aprendam uma nova linguagem, enquanto ZK VMs são compatíveis com ferramentas e linguagens convencionais já em uso. Uma vantagem chave das DSLs é que a prova ZK ou circuito gerado é agnóstico à cadeia subjacente, o que facilita seu uso em qualquer cadeia, desde que escrevamos um verificador para essa cadeia. As ZK VMs, no entanto, têm algum risco de **dependência de fornecedor** (*vendor lock-in*).

**Privacidade usando conhecimento zero**

Quando pensamos em privacidade, a primeira coisa que vem à mente é **criptografia**, e podemos imaginar várias maneiras de alcançar privacidade, as quais descrevemos a seguir:

1. **Podemos possivelmente criptografar transações** para que os dados sejam confidenciais, mas essa abordagem é impraticável por dois motivos:
   * Primeiro, a verificação pública das transações não é possível, já que os valores estão ocultos.
   * Segundo, mesmo que algum esquema de criptografia seja utilizado, o gerenciamento de chaves se torna um grande problema devido ao grande número de chaves exigidas.
2. A **segunda abordagem** é o uso de **esquemas de compromisso**, que são uma solução bastante promissora. Podemos substituir o valor/montante da transação por um compromisso com o valor, depois nos comprometemos com esse valor e o publicamos na blockchain.
   * Novamente, o problema é que, como o valor está oculto, a transação não é verificável publicamente.
   * Além disso, é possível comprometer-se com um valor negativo.
3. A **solução** é anexar **provas de conhecimento zero (ZKPs)** ao compromisso, que provam duas coisas:
   * **Compromisso de intervalo** (*range commitment*), que garante que o número é positivo e está dentro de um intervalo válido.
   * **Compromisso de valor** (*value commitment*), que oculta o valor real da transação.  
     Compromissos de intervalo são mais difíceis de fazer, enquanto os de valor são relativamente mais simples, pois se traduzem em uma equação linear simples, mais fácil de lidar.

Vamos agora explorar **compromissos criptográficos**, um primitivo criptográfico essencial para construir protocolos de conhecimento zero no blockchain.

**Compromissos criptográficos**

Um **esquema de compromisso criptográfico** é um protocolo em duas fases composto por dois algoritmos: um algoritmo de compromisso e um algoritmo de verificação:

* Commit(m, r) -> c
* Verify(m, c, r) -> aceitar ou rejeitar

Na primeira fase, chamada de **fase de compromisso**, o remetente envia ao receptor uma string de compromisso c.  
Alice executa o algoritmo commit para se comprometer com uma mensagem m usando uma aleatoriedade secreta r e produz uma string de compromisso c. Essa string c é enviada para Bob.

Na segunda fase, chamada de **fase de abertura**, o receptor abre o compromisso para verificar se o remetente realmente se comprometeu com a mensagem m e não trapaceou.

Isso é verificado usando o algoritmo verify. Esse algoritmo recebe três valores — mensagem m, compromisso c e aleatoriedade r — e, com base na saída, o verificador **aceita ou rejeita** o compromisso.  
Uma vez que o compromisso é aberto, qualquer pessoa pode verificar publicamente se ele foi aberto corretamente.

Podemos pensar em compromissos criptográficos como um **análogo digital** ou equivalente criptográfico de envelopes lacrados.  
Há duas propriedades de segurança dos compromissos criptográficos: **ocultação** (*hiding*) e **vinculação** (*binding*).

* A propriedade de ocultação garante que a mensagem comprometida não seja revelada antes da fase de abertura.  
  Em outras palavras, o compromisso c não revela nada sobre a mensagem.
* A propriedade de vinculação garante que, uma vez que o valor seja comprometido, ele não possa ser alterado depois.  
  Em outras palavras, uma vez que o remetente se comprometeu com um valor, ele **não pode** abrir o compromisso com uma mensagem diferente m que o algoritmo verify também aceite.

Existem dois tipos de esquemas de compromisso com base na força das propriedades de segurança:

* **Esquemas de compromisso padrão**, que protegem contra um receptor em tempo polinomial probabilístico (*PPT*) e um remetente com poder ilimitado. Tais esquemas são **computacionalmente ocultos** e **informacionalmente vinculantes**.
* **Esquemas de compromisso perfeitos**, que protegem contra um remetente PPT e um receptor com poder ilimitado. Tais esquemas são **informacionalmente ocultos** e **computacionalmente vinculantes**.

Esquemas de compromisso que são **simultaneamente informacionalmente ocultos e vinculantes não existem**.

Há vários modelos de esquemas de compromisso, incluindo esquemas baseados em hash, compromissos de Pedersen e esquemas de compromisso polinomial. Um tipo de esquema de compromisso chamado **compromisso de Pedersen** é muito útil devido às suas propriedades homomórficas. Vamos primeiro ver como o compromisso de Pedersen é construído e depois veremos como sua propriedade homomórfica funciona.

O esquema funciona em três fases: configuração pública (*setup*), compromisso (*commit*) e abertura (*open*).

**Fase de configuração (Setup):**

1. Seja 𝔾 um grupo cíclico finito de ordem prima.
2. Obtenha dois elementos g,h∈𝔾g, h \in 𝔾g,h∈G, onde
   * ggg é um elemento fixo conhecido do grupo (chamado de gerador),
   * e hhh é um elemento aleatório uniformemente escolhido de 𝔾.
3. Seja qqq a ordem de 𝔾, que é exponencialmente grande.

**Fase de compromisso (Commit):**

1. O comprometedor compromete-se com um número m∈Zqm \in \mathbb{Z}\_qm∈Zq​.
2. O comprometedor escolhe um inteiro aleatório z∈Zqz \in \mathbb{Z}\_qz∈Zq​.
3. Envia o compromisso ccc ao receptor, onde c=gm⋅hzc = g^m \cdot h^zc=gm⋅hz.

**Fase de abertura (Open):**

1. Para abrir o compromisso ccc, o comprometedor envia (m,z)(m, z)(m,z).
2. O receptor verifica se c=gm⋅hzc = g^m \cdot h^zc=gm⋅hz.

O compromisso de Pedersen é **perfeitamente oculto** e **computacionalmente vinculante**. Compromissos de Pedersen são **homomórficos aditivos**. Suponha que tenhamos duas strings de compromisso, c1c\_1c1​ e c2c\_2c2​, que são compromissos para as mensagens m1m\_1m1​ e m2m\_2m2​, respectivamente. Se multiplicarmos esses dois compromissos, o resultado será um novo compromisso c3c\_3c3​ para a soma m3m\_3m3​ de m1m\_1m1​ e m2m\_2m2​. Isso significa que, sem conhecer os valores originais m1m\_1m1​ e m2m\_2m2​ (as mensagens comprometidas originalmente), o receptor pode construir um compromisso para a **soma dos valores comprometidos**, ou seja, um compromisso c3c\_3c3​ para m3m\_3m3​, onde m3=m1+m2m\_3 = m\_1 + m\_2m3​=m1​+m2​, e c3c\_3c3​ pode ser aberto como m3m\_3m3​ usando as informações de abertura correspondentes.

Um adversário de tempo polinomial probabilístico (**PPT**) tem acesso a um algoritmo aleatório de tempo polinomial para quebrar a segurança do sistema. Podemos pensar nele como uma entidade com recursos computacionais limitados, capaz de quebrar o sistema apenas se tiver acesso a um algoritmo PPT.  
Algoritmos de tempo polinomial são aqueles que rodam em um tempo “razoável” em função do tamanho da entrada. Mais formalmente, um algoritmo é de tempo polinomial se executa em O(nc)O(n^c)O(nc), onde ccc é uma constante. Um algoritmo PPT é um algoritmo de tempo polinomial **randomizado**, ou seja, que possui acesso a uma fonte de aleatoriedade (por exemplo, lançar moedas).

Compromissos de Pedersen são comumente usados, com algumas modificações, em vários blockchains que preservam a privacidade, incluindo **Monero** e **Zcash**. Observe que **compromissos de Pedersen, sozinhos, não conseguem criar uma solução de privacidade completa**, mas servem como um **componente** em uma solução maior que inclui **provas de conhecimento zero (ZKPs)**, que discutiremos a seguir.

**Provas de conhecimento zero (Zero-knowledge proofs)**

A primeira prova ZK foi inventada por Goldwasser, Micali e Rackoff em 1985. Argumentos sucintos transparentes a partir de PCP foram introduzidos por Kilian e Micali em 1992 e 1994, respectivamente. Essas provas são os primeiros SNARKs, mas eram impraticáveis devido ao tempo do provador; ainda assim, as provas eram curtas em tamanho e rápidas de verificar. Por muito tempo, ZKPs foram considerados impraticáveis, mas em 2013 houve um avanço que introduziu tempo de provador linear com uma prova de tamanho constante. Isso foi melhorado em 2016. Ambos os artigos estão disponíveis aqui (GGPR13 – <https://eprint.iacr.org/2012/215> e Groth16 – <https://eprint.iacr.org/2016/260>).

Em 2013, Pinocchio, o primeiro ZK SNARK prático, foi introduzido. Também em 2013 foi introduzido o tinyRAM, a primeira VM de ZK. Em 2018, o primeiro ZK rollup de Barry Whitehat, a linguagem Circom da iden3, e o primeiro ZKSTARK da Starkware foram introduzidos. Em 2019, PLONK, um marco importante, foi alcançado, o que melhorou a usabilidade dos SNARKs. Sua configuração é universal e atualizável, o que significa que a configuração precisa ser executada apenas uma vez e então pode ser usada para qualquer programa, em vez de executar uma configuração separada para cada novo programa, como era o caso nos esquemas anteriores, por exemplo, Zcash. Em segundo lugar, o PLONK usa compromisso polinomial, que é intercambiável com qualquer outro esquema de compromisso para alcançar melhorias diferentes. Isso significa que o esquema de prova e o esquema de compromisso são separados, e um esquema de compromisso diferente e mais eficiente pode ser usado. O Plonk usa KGZ10 (Kate), mas pode ser trocado por FRI ou outro esquema.

O acrônimo FRI significa Fast Reed-Solomon IOP of Proximity, e IOP significa Prova de Oráculo Interativa. O protocolo FRI assegura que um polinômio comprometido tenha grau limitado.

O FRI permitiria que ele se tornasse transparente e seguro pós-quântico, enquanto mantém o mesmo esquema de prova, ou seja, PLONK. Esquemas de prova anteriores como Marlin e Sonic eram mais complexos e caros. Em 2020, foram desenvolvidos o Cairo ZKVM e o Halo2. 2021 viu o desenvolvimento do Circom 2.0 e do Mina. Em 2022, muitos projetos foram introduzidos, incluindo o Plonky2, que é o ZKSNARK mais rápido até agora, junto com muitos outros projetos, como os ZKEVMs, incluindo Scroll e zkSync. Não há sinal de que esse desenvolvimento vá parar ou mesmo atingir um platô em breve; cada vez mais pesquisas e desenvolvimentos estão sendo realizados por equipes entusiasmadas, e espera-se que essa tendência cresça ainda mais com mais inovações no horizonte. Como agora é uma crença estabelecida que o ZK complementa muito bem as blockchains, o futuro da blockchain será fortemente orientado em torno do conhecimento zero.

<https://eprint.iacr.org/2012/215>  
<https://eprint.iacr.org/2016/260>

**Privacidade em Blockchain**

A primeira criptomoeda a implementar com sucesso ZKPs (especificamente ZK-SNARKs – Argumento de Conhecimento Sucinto Não-Interativo) para garantir privacidade na blockchain é o Zcash. A mesma ideia pode ser implementada no Ethereum usando contratos inteligentes e também em outras blockchains. O artigo de pesquisa original está disponível em: <https://eprint.iacr.org/2013/879.pdf>. Outro excelente artigo está disponível aqui: <http://chriseth.github.io/notes/articles/zksnarks/zksnarks.pdf>.

Há outro tipo de ZKP chamado argumento de conhecimento transparente sucinto de conhecimento zero (zk-STARK), que é uma melhoria sobre os zk-SNARKs no sentido de que os zk-STARKs consomem muito menos largura de banda e armazenamento. Além disso, eles não exigem a configuração inicial, algo controversa, confiável que é necessária para zk-SNARKs. Fundamentalmente, eles são SNARKs, mas sem configuração confiável. Além disso, os zk-STARKs são muito mais rápidos, pois não fazem uso de curvas elípticas nem dependem de hashes. O artigo de pesquisa original para zk-STARKs está disponível aqui: <https://eprint.iacr.org/2018/046.pdf>.

Bulletproofs são provas curtas de conhecimento zero não-interativas. Elas não requerem configuração confiável. Esse esquema permite que um provador prove que um número criptografado está dentro de um intervalo de números sem revelar nenhuma outra informação sobre o número. Mais informações sobre Bulletproofs estão disponíveis aqui: <https://electroneropulse.org/public/doc/Bulletproof%20RingCT.pdf>.

Uma rápida comparação das técnicas mencionadas acima é mostrada abaixo:

| **Tipo/Propriedade** | **Tamanho da prova** | **Tempo do provador** | **Tempo de verificação** | **Configuração confiável requerida** |
| --- | --- | --- | --- | --- |
| SNARKs | Pequeno (~288 bytes) | Médio (~2,3 segundos) | Pequeno (~10 milissegundos) | Sim |
| STARKs | Grande (~40–50 KB) | Pequeno (~1,6 segundos) | Médio (~16 segundos) | Não |
| Bulletproof | Médio (<1 KB) | Grande (~30 segundos) | Grande (~1100 segundos) | Não |

Cobrimos ZKPs no Capítulo 21 e no Capítulo 4. Aqui nos concentraremos mais em ZKPs não-interativos e cobriremos SNARKs em mais detalhes.

Geralmente, há duas classes de argumentos de conhecimento zero: protocolos sigma e SNARKs.

Note que “prova” e “argumento” são dois termos usados às vezes de forma intercambiável. Em S\*ARKs, o “AR” representa “argumentos”, indicando que não são provas. Provas de conhecimento zero, no entanto, **são** provas, e não argumentos. É importante entender essa diferença. A diferença entre um argumento e uma prova tem a ver com quão forte é a propriedade de solidez (soundness) no protocolo. No caso dos argumentos, a propriedade de solidez é segura contra um adversário (provador) limitado polinomialmente, enquanto que as provas são seguras contra um adversário (provador) com capacidade computacional ilimitada. Em outras palavras, provas são seguras teoricamente (informação teórica), enquanto que argumentos são seguros computacionalmente.

<https://eprint.iacr.org/2013/879.pdf>  
<http://chriseth.github.io/notes/articles/zksnarks/zksnarks.pdf>  
<https://eprint.iacr.org/2018/046.pdf>  
<https://electroneropulse.org/public/doc/Bulletproof%20RingCT.pdf>

Protocolos sigma são uma forma eficiente de criar ZKPs. Estes são protocolos de três rodadas com uma fase de prova, desafio e resposta com um verificador, que é capaz de lançar uma moeda pública (aleatoriedade), conforme mostrado na Figura 18.3 abaixo:



Figura 18.3: Protocolos sigma

Como os SNARKs são a tecnologia mais usada e promissora para implementar privacidade em blockchains, focaremos mais na exploração dos SNARKs.

Como vimos no Capítulo 4, ZKPs tradicionalmente têm grandes estruturas de dados e requerem múltiplas interações desafio-resposta entre um provador e um verificador para que o provador prove alguma afirmação. Protocolos interativos podem ser úteis em cenários limitados; no entanto, NIZKs são mais adequados para cenários baseados em blockchain, onde o provador pode publicar a prova online/on-chain e os verificadores podem verificar de forma independente (assíncrona). Isso é mostrado na Figura 18.4 abaixo:

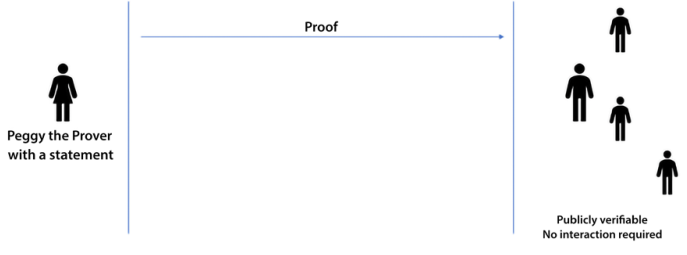


Figura 18.4: NIZK

Para alcançar a não interatividade, várias técnicas podem ser empregadas, incluindo um modelo de string de referência comum, onde os participantes compartilham uma string aleatória. Isso é crucial para provas NIZK; sem CRS, NIZK não pode existir. Falando praticamente, CRS é um conjunto de pontos de curva elíptica de uma forma específica.

Uma representação visual típica de um esquema baseado em CRS é mostrada na Figura 18.5:

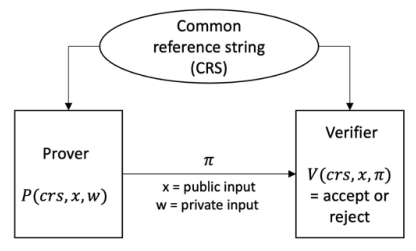


Figura 18.5: Esquema de prova baseado em CRS

As configurações confiáveis geralmente são chamadas de cerimônias de configuração, onde a chave de prova para o provador e a chave de verificação para o verificador são geradas. Aqui, também é produzido algum "lixo tóxico", que precisa ser destruído. Aqui, lixo tóxico significa simplesmente alguma aleatoriedade secreta que precisa ser destruída de forma privada; caso contrário, a segurança do sistema é comprometida.

Existem quatro métodos que podem ser usados para gerar strings de referência, a saber: confiável, subvertido, transparente e baseado em MPC:

Um modelo confiável simplesmente significa que há apenas uma parte confiável gerando a string de referência, o que claramente não é aceitável na prática, pois se essa única parte se tornar maliciosa, não há como detê-la.

A abordagem subvertida diz que mesmo se o provador tiver gerado a SRS (isto é, conhecer o segredo ou, em outras palavras, subverter totalmente o processo de configuração), ainda assim ele não será capaz de provar uma prova falsa. Isso, é claro, novamente não é prático, pois foi provado como um resultado de impossibilidade que ZK não pode ser alcançado usando esse método.

Em configurações transparentes, em vez de SRS, uma string de referência uniforme (URS) é gerada aleatoriamente e usada, o que significa que nenhum lixo tóxico (um segredo, isto é, chaves e aleatoriedade, etc.) é produzido. Note que o esquema de compromisso polinomial usado no protocolo dita inteiramente a transparência do SNARK; se o PCS usa URS, então o protocolo SNARK também usa URS. Esta opção permite configuração sem confiança.

A abordagem baseada em MPC permite que várias partes gerem conjuntamente uma string de referência estruturada de forma descentralizada sem comprometer a segurança. Desde que mesmo um único participante tenha excluído sua parte do segredo, todo o esquema pode ser considerado seguro. Computação multipartidária (MPC) é um método que permite que várias partes computem cooperativamente uma função sobre suas entradas privadas, sem revelar essas entradas entre si. MPC permite que várias partes colaborem e executem um cálculo mantendo suas entradas e resultados intermediários em segredo.

A configuração confiável foi realizada no Zcash, Aztec e vários outros protocolos. A ressalva aqui é que, se a SRS não for universal, então essa cerimônia de configuração precisa ser executada toda vez que os requisitos do circuito mudam, por exemplo, um novo tipo de transação ou mudanças na estrutura da transação existente. É por isso que o Zcash teve que executar a cerimônia novamente em 2018 (a cerimônia Sapling), após a primeira vez em 2016 (a cerimônia Sprout).

Também podemos dividir os SNARKs em três tipos com base nos requisitos de configuração. Existem três tipos principais de configurações:

* **SNARKs baseados em configuração confiável não universal (propósito específico – configuração confiável por circuito)**, nos quais eles são construídos para um único circuito. Eles têm uma grande sobrecarga de uma string de referência comum grande. Aqui, o procedimento de configuração gera alguns dados aleatórios que devem ser mantidos em segredo; caso contrário, o provador será capaz de provar declarações falsas. Uma vez que a configuração é feita, esses dados são destruídos. Eles foram usados primeiro no Zcash; isso é aceitável mesmo com as limitações, porque a configuração não precisa ser atualizada, já que há apenas um tipo de circuito, ou seja, uma transferência de transação. A configuração deve ser feita novamente se o tipo de circuito mudar.
* **Uma configuração confiável universal**, onde o SNARK requer apenas uma configuração confiável uma vez. Eles têm uma string de referência estruturada (SRS), que é atualizável e menor que a CRS. Aqui, a string de referência tem alguma estrutura baseada em aleatoriedade secreta. Esta também é uma configuração confiável, mas é universal e atualizável porque o material secreto (aleatoriedade, armadilha, etc.) nessa configuração é independente do circuito. Isso pode ser pensado como um processo de duas etapas onde um procedimento inicial único é executado em segredo para gerar uma string de configuração atualizável e quaisquer segredos são destruídos. No entanto, agora essa string de configuração é atualizável para qualquer novo circuito e pode pré-processar quaisquer novos circuitos sem depender de dados secretos.
* **O terceiro tipo são os SNARKs transparentes ou STARKs**, que não requerem configuração confiável, pois não usam dados secretos.

Como os SNARKs produzem provas pequenas (sucintas) que são rápidas de verificar, essas provas podem ser usadas não apenas para privacidade, mas também para escalabilidade, como vimos no Capítulo 17, Escalabilidade, porque são curtas em tamanho, e os verificadores só precisam processar e verificar um pequeno pedaço de dados em vez de todas as transações. Essas provas, embora eficientes, têm dois problemas principais. Primeiro, leva muito tempo para gerar provas e, segundo, os SNARKs requerem SRSs, o que significa confiar em terceiros.

Vamos agora ver como os ZK-SNARKs são construídos.

**Construindo ZK-SNARKs**

De modo geral, gerar um SNARK é um processo de duas etapas. A primeira etapa, que envolve a “interface” do sistema, é converter um programa em um modelo equivalente que possa ser verificado de forma probabilística.

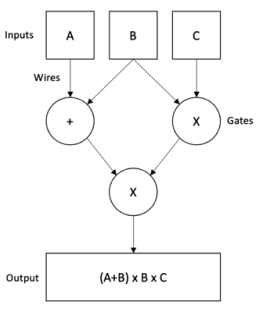
O Zcash lançou o protocolo de pagamento protegido Orchard, que usa o sistema de prova Halo 2 e elimina a necessidade de uma string de referência estruturada e, assim, das cerimônias de configuração confiável. Isso significa que futuras atualizações de circuitos podem ser feitas sem uma configuração confiável. Mais informações sobre isso estão disponíveis aqui: <https://zips.z.cash/zip-0224>.

Isso significa transformar o programa a ser provado em um circuito aritmético, ou seja, geralmente em um problema de satisfatibilidade de circuito. O circuito aritmético é representado ainda por um sistema de restrição aritmética, que pode ser R1CS – o sistema de restrição de posto 1 (Rank 1 Constraint System) ou Relação de Produto de Hadamard (HPR – Hadamard Product Relation). Em um nível alto, o sistema de restrição codifica os circuitos como matrizes. A primeira etapa pode ser vista como um processo onde o programa é transformado em um circuito equivalente, o que basicamente nos permite tornar a verificação da prova por um verificador não interativa. O código do programa é transformado em um circuito aritmético, que é um circuito algébrico que representa o código do programa. Praticamente, o circuito aritmético é descrito como matrizes e operações relevantes em álgebra linear. O provador e o verificador executam o SNARK para a satisfatibilidade do circuito. Aqui geralmente são usadas DSLs como Circom e Zokrates para escrever o circuito. Circom é uma linguagem de nível mais baixo onde o programa inteiro é escrito quase porta por porta; Zokrates é um pouco mais de alto nível do que isso e nos permite criar programas em um estilo de código mais familiar ao Python. Cairo e Noir são outras duas linguagens que podem ser usadas para escrever circuitos para representar o programa. Essa abordagem de usar uma DSL para escrever programas é muito mais fácil do que escrever um compilador para pegar um programa de alto nível escrito em Rust ou C e converter isso diretamente em um circuito equivalente. Tal abordagem não é muito prática; portanto, DSLs específicas são usadas para escrever circuitos.

Qualquer programa que o provador deseja provar é primeiro convertido em um circuito aritmético, o que também é chamado de achatamento (flattening). Um circuito aritmético é uma construção que representa uma função matemática como um grafo acíclico direcionado (DAG) consistindo de portas, entradas, fios e saídas. As portas são os vértices e os fios são as arestas. Cada porta no circuito executa uma operação aritmética específica (como adição ou multiplicação) sobre suas entradas e produz uma saída, e as saídas das portas são usadas como entradas para outras portas até que a saída final seja produzida. Para transportar valores, são usados fios (arestas). Os nós de entrada são rotulados com variáveis e constantes.

Para construir ZK-SNARKs, os circuitos aritméticos são usados para representar o cálculo que o provador deseja provar. Mais precisamente, o provador tenta convencer o verificador de que possui uma solução para o circuito aritmético.

Tal circuito é mostrado na Figura 18.6 abaixo, que computa a expressão (a + b) × b × c:



**Figura 18.6: Um circuito aritmético**

Polinômios são amplamente usados para representar muitos problemas em criptografia. Circuitos também são definidos sobre um campo finito primo como polinômios multivariados, e a estrutura do circuito fornece um modelo computacional para avaliar o polinômio.

A característica chave dos ZKPs baseados em circuitos aritméticos é que eles permitem que um provador produza uma prova que mostra que o provador conhece um valor de entrada para o circuito, o qual produz uma saída predefinida, enquanto mantém todas ou algumas entradas em segredo. Esses sistemas de prova asseguram que um provador não seja capaz de gerar uma prova válida sem o conhecimento das entradas corretas para o circuito e que as entradas privadas sejam ofuscadas, de forma que nenhuma informação seja vazada sobre essas entradas privadas.

Vimos anteriormente o que são um argumento e uma prova. Vamos agora definir o sistema de argumento ZK à luz do que aprendemos sobre circuitos. Em um sistema de argumento temos duas partes, um verificador e um provador. Imagine que temos um circuito que recebe duas entradas, x e w. x é a entrada pública e w é a testemunha secreta que o provador está tentando provar que conhece. O objetivo do provador é convencer o verificador de que existe uma testemunha w, que quando processada pelo circuito junto com a entrada pública x produz a saída desejada. O verificador não sabe o que é w e conhece apenas x, a entrada pública. O verificador e o provador participam de várias rodadas de troca de mensagens, e ao final do protocolo o verificador ou rejeita ou aceita a prova. Em outras palavras, o verificador está convencido de que o provador de fato conhece x, ou não.

Um sistema de argumento de conhecimento zero não interativo tem quatro propriedades: completude, argumento de conhecimento, conhecimento zero e sucintez:

* **Completude** significa que um provador honesto é sempre capaz de convencer um verificador honesto de que a prova está correta.
* **Argumento de conhecimento** significa que, se o verificador aceitou a prova, então o provador indubitavelmente conhece uma informação secreta, chamada de testemunha, que satisfaz o circuito. Em outras palavras, a prova válida só pode ser gerada pelo provador que de fato conhece a entrada privada. Mais precisamente, isso significa que o provador conhece a testemunha se ela puder ser extraída do provador. A extração significa que existe um procedimento eficiente que pode extrair os segredos de um provador. Um sistema de argumento é computacionalmente sólido, em vez de teoricamente sólido, onde a suposição é que é impossível provar uma afirmação falsa ao provador. Em um sistema de argumento, dizemos que é **inviável** enganar o provador em vez de impossível. Se assumirmos como impossível, então é uma **prova** em vez de um **argumento**.
* **Conhecimento zero** significa que a prova produzida pelos provadores não revela absolutamente nada sobre a testemunha. Em outras palavras, ela não revela nada além de se a afirmação feita é verdadeira ou falsa. Formalmente, essa propriedade significa que se o verificador for capaz de regenerar a prova por conta própria, ele não aprenderá nada novo; a prova ainda não revelará nada sobre a testemunha. Podemos dizer que um sistema de argumento ZK é de conhecimento zero para um circuito se existir um procedimento eficiente, chamado de simulador, que possa gerar os parâmetros públicos e a prova, os quais são indistinguíveis dos parâmetros públicos reais gerados pelo procedimento de configuração e da prova gerada pelo provador real, apenas usando a entrada pública x, ou seja, sem conhecer a testemunha.
* **Sucintez** significa que a prova produzida é pequena em tamanho e é rapidamente verificável pelo verificador. A aceleração no tempo de verificação é, de fato, alcançada por causa da fase de configuração, onde os parâmetros públicos são gerados.

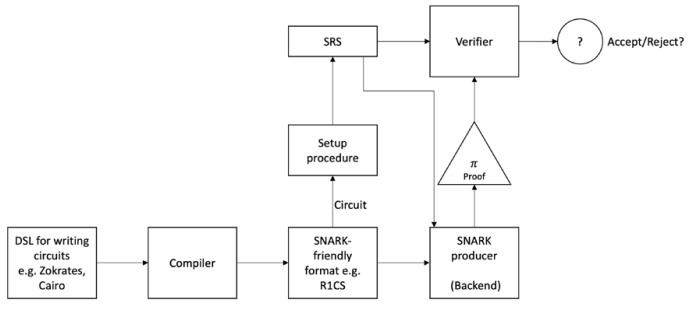
Podemos pensar na fase de configuração como o circuito sendo “pré-processado” de tal forma que o verificador não precise ler o circuito inteiro. O provador, é claro, precisa ler completamente, mas o verificador pode evitá-lo confiando essencialmente em uma “versão curta” do circuito produzida devido ao pré-processamento realizado na fase de configuração.

Após o circuito ser gerado, os parâmetros públicos podem ser gerados pelo procedimento de configuração, o qual pega o circuito aritmético e produz os parâmetros necessários. Esses parâmetros públicos são usados pelo provador junto com a entrada pública x e a testemunha secreta w para produzir a prova (geralmente denotada por π). O verificador também usa esses parâmetros públicos junto com a entrada pública x para verificar a prova enviada pelo provador.

SNARKs dependem de uma configuração confiável onde uma string de referência comum é gerada para tornar os tamanhos das provas menores. Temos vários tipos de strings de referência, incluindo CRS, SRS e URS. A SRS é essencialmente uma chave de prova e geralmente é tão grande quanto o circuito. Existe uma função armadilha (trapdoor) nessa SRS, e se o provador a descobrir, ele poderá facilmente criar declarações falsas que conseguirá provar ao verificador como verdadeiras. Essa armadilha é, de fato, conhecida pela entidade que gera essa SRS, e essa entidade precisa ser confiável pelo sistema. A expectativa é que essa parte destruirá esse “lixo tóxico” após a configuração da SRS. Isso, é claro, não é ideal, e muitas pesquisas têm sido realizadas para minimizar os requisitos de confiança. Um exemplo-chave é criar a SRS através do protocolo MPC. No entanto, não há necessidade de uma configuração SRS nos STARKs. A SRS para circuitos simples é gerenciável, mas circuitos complexos com possivelmente bilhões e trilhões de portas podem se tornar um problema para gerenciar em termos de armazenamento e distribuição para os provadores.

Na segunda etapa, envolvendo o backend do sistema, um esquema de prova é aplicado para a satisfatibilidade do circuito. Alguns exemplos dos esquemas de prova incluem Groth16, GM17, Marlin, PLONK, Libra, Supersonic, Hyrax, entre muitos outros. O principal requisito por trás do desenvolvimento do backend é criar um SNARK que tanto o provador quanto o verificador possam executar para satisfazer o circuito. Espera-se que o esquema de prova tenha tempos rápidos de verificação pelo verificador, seja sucinto (pequeno em tamanho – alguns bytes ou KBs), e seja pelo menos linear (em termos de complexidade algorítmica) ao tamanho do programa em relação ao tempo do provador.

Podemos visualizar a interface, configuração e backend como um sistema ZK-SNARK na Figura 18.7 abaixo:



**Figura 18.7: Um sistema ZK-SNARK típico**

O design da segunda etapa, ou seja, o backend ou esquema de prova, geralmente inclui três etapas:

* Criar (ou usar) uma Prova de Oráculo Interativa Polinomial (PIOP) para satisfatibilidade de circuito. Em outras palavras, criar um esquema de prova que demonstre que a afirmação (programa) para a qual o circuito foi construído é verdadeira.
* Criar (ou usar) um esquema de compromisso polinomial para construir um argumento interativo sucinto.
* Usar a heurística de Fiat-Shamir para transformar esse argumento interativo em um protocolo não interativo. Isso é alcançado substituindo a aleatoriedade do verificador por um oráculo aleatório, que é instanciado por uma função de hash criptográfica. O provador é incapaz de prever o resultado da função de hash, fornecendo assim a aleatoriedade necessária para atingir a propriedade de solidez do ZKP.

Combinando as duas primeiras etapas no “backend” produz-se um SNARK, ou seja, PIOP + PCS = SNARK.

Usamos o termo IOP bastante até aqui. Vamos ver o que isso significa. IOP significa Prova de Oráculo Interativa (*Interactive Oracle Proof*). Trata-se de uma prova interativa no sentido de que o verificador não precisa ler a mensagem inteira do provador; em vez disso, o verificador tem acesso a um oráculo para a mensagem do provador e pode consultá-la de forma probabilística. Na prática, isso significa que o verificador consulta alguns pontos de avaliação em vez de ler toda a mensagem. Aqui, o provador possui uma entrada pública e uma testemunha, enquanto o verificador tem apenas a entrada pública. As mensagens do provador são algumas cadeias de bits arbitrárias (isto é, codificações polinomiais), enquanto as mensagens do verificador são algumas aleatoriedades (isto é, consultas sobre pontos de avaliação). O provador e o verificador participam de várias rodadas dessas mensagens e, eventualmente, o verificador gera consultas para interrogar o oráculo em locais específicos (pontos de avaliação).

Uma vez que o resultado dessas consultas é gerado, o verificador decide se aceita ou rejeita a prova. Para garantir que as mensagens sejam de fato polinômios, é realizado um teste chamado “teste de baixo grau” (*low-degree test*), que verifica o grau do polinômio. No entanto, isso produz provas de tamanho grande e requer várias rodadas para operar, o que é bastante custoso. Entretanto, se pudermos assumir que as mensagens enviadas do provador ao verificador são polinômios sobre um campo finito, então não há necessidade do teste de baixo grau, e isso resultará em menor custo. Isso é exatamente o que os IOPs polinomiais fazem quando as mensagens do provador são polinômios e, por causa dessa suposição, não há teste de baixo grau, o que reduz o custo e melhora a eficiência. IOPs podem ser categorizados em três classes com base em como evoluíram ao longo do tempo:

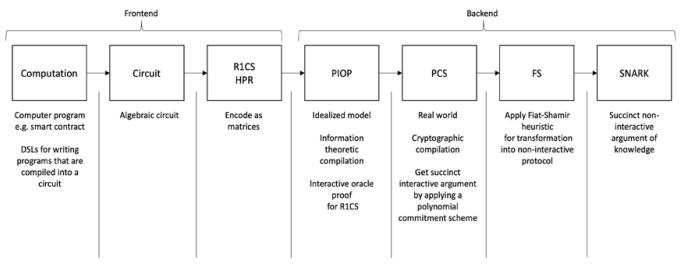
* **Provas interativas**, que são uma categoria padrão onde um único provador e um verificador participam de um protocolo para provar a veracidade de uma afirmação do provador. Alguns exemplos de SNARKs nesta categoria são Libra e Hyrax. O tempo do provador é o mais baixo nessa categoria, mas o tamanho da prova é grande e o tempo do verificador é alto.
* **Provas interativas com múltiplos provadores**, onde, em vez de um único provador, existem múltiplos provadores. Alguns exemplos-chave nessa categoria são Spartan e Brakedown.
* **PIOPs de rodada constante**, que são IOPs com complexidade de rodada constante. Exemplos importantes são Marlin e PLONK. O tempo do provador é alto nesta categoria; no entanto, o tamanho da prova e o tempo de verificação são rápidos.

Após um PIOP ser criado, é hora de usar um esquema de compromisso polinomial existente ou criar um novo para construir um argumento interativo sucinto. Esquemas de compromisso polinomial permitem que um comprometedor (provador) se comprometa com um polinômio enviando uma string curta ao verificador.

Mais tarde, o verificador pode pedir ao provador que avalie um único ponto do polinômio. O provador responde com o resultado e uma prova, que o verificador avalia. O objetivo aqui é que o provador **não** seja capaz de produzir uma prova convincente de uma avaliação incorreta do ponto no polinômio. Em segundo lugar, o compromisso e a prova devem ser fáceis de gerar e pequenos, e a prova deve ser fácil e rápida de verificar. Os esquemas de compromisso polinomial podem ser classificados em quatro categorias de acordo com as suposições criptográficas em que se baseiam:

* **IOP com uma função de hash criptográfica resistente a colisões**, que não requer configuração (ou seja, é transparente) e é resistente a ataques quânticos. Os exemplos mais comuns desse esquema incluem **FRI** (<https://drops.dagstuhl.de/opus/volltexte/2018/9018/pdf/LIPIcs-ICALP-2018-14.pdf>) e compromissos **Ligero**.
* **Baseado em ECDLP**, que usa curvas elípticas e é transparente, mas **não é resistente a ataques quânticos**. **Bulletproofs** (<https://eprint.iacr.org/2017/1066.pdf>) são um exemplo comum desse tipo de esquema de compromisso.
* **Baseado em grupos com emparelhamento**, que requer configuração confiável e **também não é resistente a ataques quânticos**. Um exemplo comum é o compromisso **KZG10** (<https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf>). Outro exemplo é o **Dory**.
* **Baseado em grupos de ordem desconhecida**, que são transparentes se usarem grupos de classes, mas **não são resistentes a ataques quânticos**. Uma desvantagem importante é que o tempo do provador é muito lento, devido ao uso de grupos de classes. Um exemplo comum de tais esquemas é o **DARK** (<https://eprint.iacr.org/2019/1229.pdf>).

Podemos visualizar um pipeline genérico de construção de SNARKs com base no que discutimos até agora na Figura 18.8. Também podemos ver isso como um processo de como vamos de um programa para um SNARK:

  
**Figura 18.8: Pipeline genérico de construção de um SNARK**

Em resumo, o pipeline de construção começa com a aritmetização de um cálculo onde técnicas como R1CS e HPR são usadas. A próxima etapa é a compilação dessa representação aritmética em um protocolo de natureza teórica da informação, onde modelos idealizados como IOPs polinomiais (PIOPs), PCPs e PCPs lineares baseados em oráculos são usados.

<https://drops.dagstuhl.de/opus/volltexte/2018/9018/pdf/LIPIcs-ICALP-2018-14.pdf>  
<https://eprint.iacr.org/2017/1066.pdf>  
<https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf>  
<https://eprint.iacr.org/2019/1229.pdf>

Uma vez que estejamos satisfeitos com a construção teórica da informação, para torná-la prática e real, substituímos o modelo teórico da informação por construções criptográficas chamadas de compromissos polinomiais. Aqui, essencialmente compilamos IOP para SNARK. Um candidato usual para PCS é o esquema de compromisso de Kate (<https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf>). É aqui que finalmente o SNARK é produzido. Podemos pensar nisso como a combinação de PIOP com PCS para gerar um esquema SNARK.

Um esquema de compromisso polinomial permite que um comprometedor se comprometa com um polinômio. As propriedades de ocultamento e vinculação aplicam-se aos esquemas de compromisso polinomial assim como em outros esquemas de compromisso. Eles também são compostos pelas fases de compromisso (*commit*) e abertura (*open*), como outros esquemas. No entanto, há uma propriedade adicional que podemos chamar de “abertura seletiva”. Essa propriedade significa que o comprometedor é capaz de abrir certas avaliações do compromisso sem revelar o polinômio inteiro. Em outras palavras, o PCS permite o comprometimento a um polinômio com uma string curta, a qual pode ser usada pelo receptor para verificar as avaliações reivindicadas do polinômio comprometido. Isso significa que o provador pode provar que possui um polinômio que satisfaz algumas propriedades exigidas **sem revelar o polinômio**.

Vamos agora ver o que é o Kate PCS e como ele funciona. O esquema pode ser dividido em três fases: configuração, compromisso, e abertura e verificação, assim como outros esquemas de compromisso.

**Configuração**

Esta é a configuração confiável única para gerar a SRS:

* Seja *e* um emparelhamento bilinear entre grupos 𝔾\_t de ordem prima *p*.
* Seja *g* um gerador. 𝔾 é um grupo de curva elíptica com suporte a emparelhamento.
* Seja *l* o grau máximo, ou seja, o limite superior do grau dos polinômios com os quais queremos nos comprometer.
* Escolha um inteiro aleatório τ do campo primo, ou seja, τ ∈ ℤ\_p = {1, ..., p−1}.
* Calcule a SRS como (g, g^τ, g^τ², ..., g^τ^l). Essa SRS consiste em codificações em 𝔾 de todas as potências de τ.
* Note que τ é o “lixo tóxico”, ou seja, o parâmetro secreto (ou armadilha) da cerimônia de configuração, o qual deve ser destruído uma vez que a SRS é gerada.

**Compromisso**

Nesta fase, é feito o compromisso com um polinômio φ sobre ℤ\_p:

* Um polinômio φ(x) = Σ φᵢ xⁱ.
* Computa-se o compromisso c = g^φ(τ). Aqui, mesmo que τ tenha sido destruído, o comprometedor ainda pode computar isso usando a SRS e a homomorfia aditiva.  
  Recordando que nossa SRS é (g, g^τ, g^τ², ..., g^τ^l).  
  Se φ(x) = Σ φᵢ xⁱ, então g^φ(τ) = ∏ (g^τⁱ)^φᵢ para todo i, o que significa que, dados os valores g^τⁱ para todos os i, o compromisso *c* pode ser computado sem o conhecimento de τ.

Em termos mais simples, isso significa que avaliamos o polinômio que queremos comprometer no ponto τ e multiplicamos isso por *g*, o que produzirá um único elemento de 𝔾, e esse será nosso compromisso *c*.  
Uma propriedade incrível a se notar aqui é que o polinômio pode ter um grau arbitrariamente grande, mas o compromisso produzido ainda será apenas um único elemento, ou seja, uma peça de dado de tamanho muito curto, geralmente cerca de ~64 bytes.

<https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf>

**Abertura**

É aqui que provamos uma avaliação. Para abrir o compromisso na entrada *a* para algum valor *b*, ou seja:

* Dada uma avaliação φ(a) = b
* O comprometedor computa e fornece a prova π.  
  Isso é basicamente um cálculo de polinômio testemunha.
* q(x) = (φ(x) – b)/(x – a) é chamado de polinômio quociente (*quotient polynomial*).  
  Tal q(x) só pode existir se φ(a) = b.  
  Se esse polinômio quociente existe, então isso constitui uma prova de avaliação.  
  Matematicamente, verifica-se se o teorema do resto do polinômio é válido.

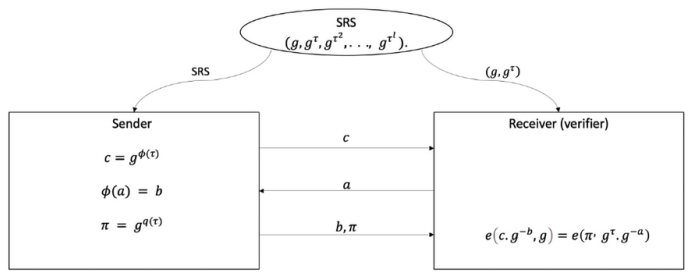
**Verificação**

Aqui é onde a prova de avaliação é verificada.  
Uma propriedade chave aqui é que a verificação pode ser feita em tempo constante:

* Dado um compromisso c = g^φ(τ), uma avaliação φ(a) = b, e a prova π
* O verificador checa se:  
  **e(c – g^b, g) = e(π, g^{τ – a})**,  
  onde *e* é um mapeamento bilinear não trivial.

Note que o verificador conhece o compromisso *c*, a avaliação *b*, a prova *π*, a entrada *a* e *g^τ*.  
Os valores *c*, *b* e *π* são fornecidos pelo provador.  
A entrada *a* é a consulta de abertura, fornecida pelo verificador.  
*g^τ* é uma entrada da SRS.  
Para a verificação, apenas *g^τ* e *g* são necessários.  
É por isso que a SRS às vezes é chamada de **chave de prova**, e *(g, g^τ)* são chamadas de **chave de verificação**.  
O verificador precisa apenas da chave de verificação para verificar as provas, e o provador usa toda a SRS como chave de prova para gerar as provas.

Podemos visualizar o PCS de Kate na Figura 18.9:



**Figura 18.9: Uma visão de alto nível do PCS KGZ**

Se temos grandes quantidades de dados, como podemos nos comprometer com eles de tal maneira que uma mensagem curta represente esse conjunto inteiro de dados?  
Em outras palavras, deveríamos ser capazes de nos comprometer de forma sucinta com esses dados.  
Em segundo lugar, deveríamos então ser capazes de provar valores em locais específicos.

É aí que o KZG é útil; podemos representar grandes dados como um polinômio e então avaliá-lo em pontos secretos. Agora você pode ver como PCS pode ser útil em blockchain. Como o compromisso de um polinômio de grau arbitrariamente grande pode ser representado por apenas um elemento de grupo, podemos simplesmente publicar esse compromisso na blockchain sem precisar publicar tudo na blockchain, economizando assim armazenamento, largura de banda e custos de gás. Em um sistema rollup, uma computação (por exemplo, uma transição de estado) pode ser representada como polinômios aos quais o provador se compromete e publica na cadeia para verificação. O verificador pode solicitar ao comprometedor (por exemplo, o provedor do rollup) uma avaliação em alguns pontos aleatórios; se as avaliações forem corretas, então significa que toda a computação está correta.

Ok, então agora, após aplicar o esquema de compromisso, alcançamos um protocolo que nos permite nos comprometer com um polinômio com uma string curta. Isso significa que é um protocolo sucinto, mas ainda é interativo.

Para torná-lo não interativo, podemos usar a heurística de Fiat-Shamir (FS) para transformar esse protocolo interativo em um não interativo. A transformação FS é uma técnica comumente usada para transformar um protocolo interativo em um protocolo não interativo. A ideia por trás da transformação FS é que o provador gera bits aleatórios por conta própria. Protocolos interativos têm um verificador que gera bits aleatórios e os envia ao provador, mas esquemas de prova não interativos transformados via FS permitem que o provador gere os bits aleatórios por conta própria. O requisito aqui é fornecer aleatoriedade uniforme suficiente para satisfazer a propriedade de solidez dos ZKPs. A interação é substituída por acesso a um oráculo aleatório não interativo. Na prática, isso é instanciado usando uma função de hash criptográfica com entradas cuidadosamente escolhidas. Após aplicar a transformação FS, finalmente obtemos um protocolo sucinto e não interativo, ou seja, um SNARK.

Existem diferentes técnicas para construir o backend de um SNARK, com diferentes prós e contras. Por exemplo, SNARKs baseados em PCP linear têm os benefícios de prova mais curta e tempos de verificação mais rápidos. No entanto, eles exigem uma configuração específica por circuito, têm tempos de provador lentos e não são resistentes a ataques quânticos. Um exemplo chave desse tipo é o **Groth16**. IOPs polinomiais combinados com PCS KZG produzem SNARKs que têm a vantagem de uma configuração universal confiável; no entanto, as provas são maiores, o tempo do provador também é mais lento, e também não são resistentes a ataques quânticos. Alguns exemplos nessa categoria são **PLONK** e **Marlin**. PCP linear e PIOPs de rodada constante produzem SNARKs que requerem configuração confiável.  
Combinar um PIOP com PCS FRI produz provas que são as menores e resistentes a ataques quânticos; no entanto, o tempo do provador é lento e o tamanho da prova ainda é bastante grande.

É assim que um SNARK é criado em alto nível. ZKPs é um assunto vasto e nem tudo pode ser coberto em um único capítulo. Para exploração adicional, um excelente recurso online é <https://zkp.science>, que pode fornecer mais informações sobre esse fascinante tema.

ZK-SNARKs têm muitas aplicações no mundo da blockchain. Eles podem ser usados para melhorar a escalabilidade da blockchain ao criar um rollup baseado em SNARK. Eles também podem ser usados para fornecer privacidade nas transações ao ofuscar os valores das transações e as informações do remetente/destinatário.

Vamos agora ver um exemplo onde vemos esses conceitos na prática.

<https://zkp.science>

**Exemplo**

Para este exemplo, usaremos o **Zokrates** (<https://zokrates.github.io/introduction.html>), que é um kit de ferramentas de código aberto que permite aos usuários construir e implantar sistemas de ZKP. Ele inclui uma linguagem de programação de alto nível chamada **Zokrates DSL**, que é projetada para escrever programas que podem ser verificados com conhecimento zero.

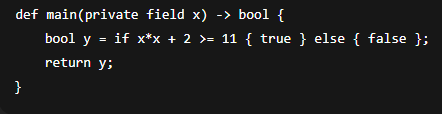
Para usar o Zokrates, programadores primeiro escrevem o programa de ZKP usando o Zokrates DSL. Ele é então compilado em um circuito aritmético de baixo nível usando o compilador do Zokrates.

Neste exemplo, o provador provará a um verificador que conhece um valor *x*, o qual, quando usado para resolver uma equação simples, resulta em uma resposta que o provador já conhece. Vamos assumir que a equação é:

y = x² + 2

A condição aqui é que se *y* for maior ou igual a 11, então é uma prova de que o provador conhece o valor de *x*, **sem revelar o valor de x**.

Então, vamos escrever um programa simples em Zokrates para isso:

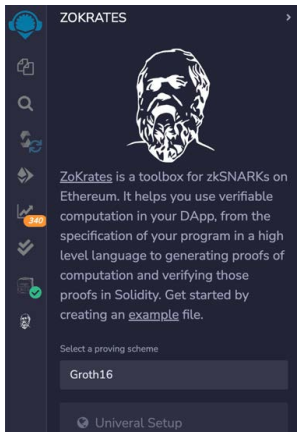


Aqui, estamos simplesmente definindo uma função principal com um campo privado *x*. Declaramos como **private** porque **não queremos revelar essa entrada**, mas ainda queremos provar que conhecemos esse valor *x*.

A seguir, temos a verificação de que se o quadrado do valor de *x* somado a 2 é maior ou igual a 11, então a variável *y*, sendo um booleano, receberá o valor **true**; caso contrário, será **false**. Na próxima instrução, retornamos *y*, que será **true** ou **false** dependendo do valor de *x*. Se o provador conhece um valor legítimo de *x* que resulta em um valor maior ou igual a 11, então ele pode provar ao verificador que de fato conhece tal valor de *x* que satisfaz a equação; caso contrário, o verificador será convencido de que o provador **não** conhece esse valor.

Usaremos o **Remix IDE** e o plugin **Zokrates** para demonstrar isso. Vamos começar:

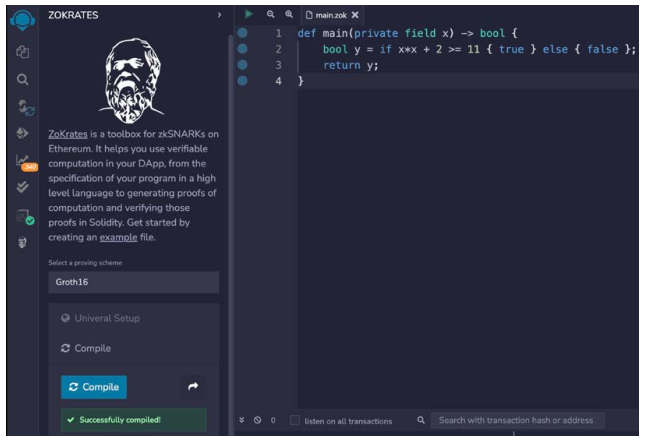
1. Primeiro, abra o Remix IDE.
2. Na página inicial, há uma visualização de plugins; nela, encontre o **Zokrates** e instale-o no Remix.  
   Uma vez instalado, você notará que ele está disponível na coluna do lado esquerdo do Remix IDE, como mostrado abaixo na **Figura 18.10**:

  
**Figura 18.10: O kit de ferramentas Zokrates no Remix IDE**

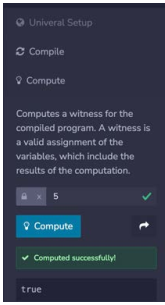
1. Uma vez que o Zokrates esteja disponível no Remix IDE, crie um novo arquivo nele e nomeie-o como **main.zok**, como mostrado na **Figura 18.11**:

  
**Figura 18.11: O arquivo main.zok**

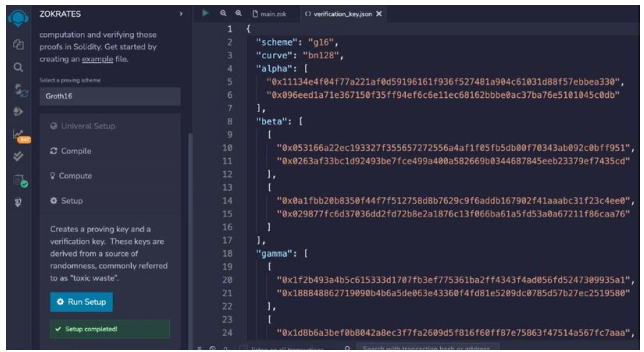
1. Uma vez que o arquivo esteja lá, clique no ícone do Zokrates no lado esquerdo e clique em **Compile**, como mostrado abaixo. Se tudo estiver correto, ele irá compilar o arquivo. Essa compilação é equivalente à criação do circuito, conforme discutido anteriormente.

  
**Figura 18.12: Compilação com o Zokrates**

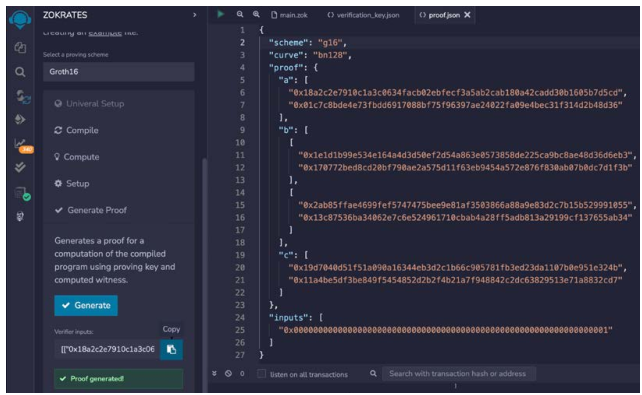
1. A próxima etapa é **calcular a testemunha** (*compute the witness*), o que pode ser feito inserindo o valor de *x* e clicando em **Compute**. Todas essas opções estão disponíveis sob o plugin do Zokrates, como mostrado abaixo.

  
**Figura 18.13: Calcular testemunha**

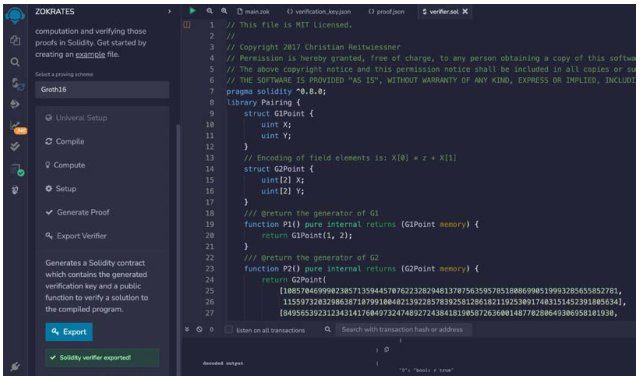
1. Agora execute a **configuração** (*setup*), que cria uma **chave de prova** (*proving key*) e uma **chave de verificação** (*verification key*), como mostrado na **Figura 18.14**:

  
**Figura 18.14: Executar a configuração**

1. A próxima etapa é **gerar a prova** usando a chave de prova e a testemunha geradas nas etapas anteriores. Para isso, clique em **Generate** sob a opção **Generate Proof** no plugin do Zokrates, como mostrado abaixo na **Figura 18.15**:

  
**Figura 18.15: Gerar prova**

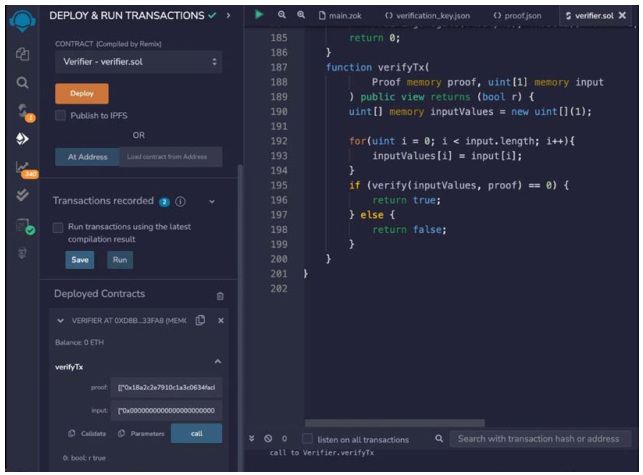
1. Uma vez que a prova seja gerada, ela criará um arquivo chamado proof.json contendo a prova. Observe que na coluna da esquerda há uma opção **Verifier inputs**; clique no botão **Copy** para manter essa prova na área de transferência ou salve-a em algum lugar, pois esta é a entrada para o contrato inteligente verificador que será gerado na próxima etapa. Usaremos essa prova mais adiante para passá-la ao contrato verificador para verificação.
2. A próxima etapa é **gerar o verificador**; clique no botão **Export** sob a opção **Export Verifier** para gerar um contrato inteligente em Solidity que contém uma função pública para verificar a prova, ou seja, a solução do programa compilado no Zokrates. Observe que esse contrato inteligente também contém a **chave de verificação**. Isso é mostrado na **Figura 18.16** abaixo:

  
**Figura 18.16: O contrato inteligente de verificação em Solidity**

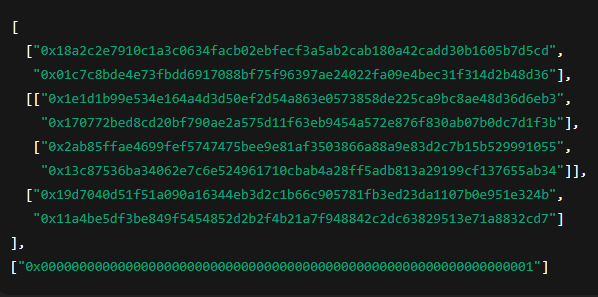
Esse arquivo de contrato se chama **verifier.sol**. Agora podemos compilá-lo e implantá-lo na blockchain, seja em uma **testnet** ou na **mainnet do Ethereum**, como aprendemos nos capítulos anteriores.  
Se você quiser implantar na rede principal, pode usar o **MetaMask** e selecionar a opção **Injected Web3** no Remix IDE, como você aprendeu antes no capítulo anterior.

Aqui, usaremos apenas a **VM do Remix** e executaremos este exemplo:

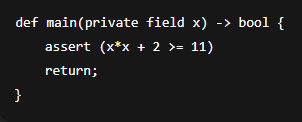
1. Agora vá até a opção **Solidity Compiler** e compile o arquivo **verifier.sol**.
2. Em **DEPLOY & RUN TRANSACTIONS**, selecione **verifier.sol** sob **CONTRACT** e clique em **Deploy**.
3. Uma vez implantado, basta invocar a função pública **verifyTx**, passando os dois parâmetros e clicando no botão **call**, como mostrado na captura de tela a seguir, na **Figura 18.17**:

  
**Figura 18.17: Invocando a função pública verifyTx do contrato inteligente verificador**

Observe que, na captura de tela acima, o valor **bool: true** foi retornado, indicando que o verificador conseguiu verificar a prova.  
Lembre-se de que **não revelamos o valor de x em momento algum** ao contrato verificador, apenas passamos dois parâmetros: o tuplo da prova e o valor da entrada, conforme mostrado abaixo:



Com isso, completamos nossa introdução ao **Zokrates** e como programas podem ser construídos e provados com conhecimento zero.  
Observe que podemos escrever o mesmo programa de outra forma, como mostrado abaixo, onde usamos a instrução assert em vez da estrutura if else:



Vou deixar isso como um exercício para o leitor, onde você pode compilar e executar esse programa usando o plugin Zokrates e o Remix IDE.

Agora que sabemos como o Zokrates funciona e como podemos construir soluções de conhecimento zero, você consegue construir um programa que **prove a um verificador, com conhecimento zero, que você tem mais de 18 anos sem revelar exatamente quantos anos tem**?  
É um exemplo simples, mas pode ser usado em vários cenários, por exemplo, provar que alguém é elegível para dirigir um carro, **sem revelar sua idade exata ou data de nascimento**.

No geral, grandes progressos foram feitos na pesquisa e desenvolvimento de provas de conhecimento zero, mas ainda existem alguns desafios.  
Por exemplo, o tema principal que está surgindo é que, **enquanto a sucintez e a universalidade estão aumentando, a velocidade do provador não está melhorando muito**.  
Hardware de alto desempenho é necessário para gerar provas em tempo razoável; esta é a razão pela qual os provedores de rollup tendem a ser máquinas potentes na nuvem. Alguns até sugeriram **ASICs e FPGAs** para esse propósito.

Se no futuro, de alguma forma, o tempo do provador puder ser reduzido enquanto se mantêm ou reduzem os tempos do verificador e os tamanhos das provas, isso melhoraria ainda mais a adoção pelo usuário e permitiria casos de uso onde hardware de consumo mais modesto, como **telefones celulares**, poderia gerar provas **sem afetar a experiência do usuário**.  
A resistência quântica é outro desafio; enquanto os STARKs são resistentes a ataques quânticos, este é outro campo interessante de pesquisa para desenvolver técnicas mais avançadas.  
Algumas pesquisas podem ser conduzidas para desenvolver **novos esquemas de compromisso polinomial que sejam ainda mais eficientes**.  
A **verificação formal** e a garantia de que as propriedades de segurança desses protocolos são corretamente projetadas e implementadas é outro assunto de grande interesse.  
Imagine um protocolo DeFi gerenciando bilhões de dólares em ativos digitais; se eu tiver uma prova formal de que o sistema de conhecimento zero implementado no DeFi funcionará corretamente em todos os casos, isso aumentará a confiança do consumidor.  
Isso também se aplica, de modo geral, não apenas a protocolos ZK, mas também a protocolos em blockchain e software, os quais discutiremos com mais detalhes no próximo capítulo, **Capítulo 19 – Segurança em Blockchain**.

**Resumo**

Este capítulo abordou a **privacidade em blockchain** – um assunto de profunda importância.  
Cobrimos o que é privacidade, seus diferentes aspectos e **como podemos resolver o problema de privacidade na blockchain**.  
Também vimos as principais tecnologias, como **ZKPs** e, particularmente, os **SNARKs**, que são a forma mais avançada e usada de ZKPs para resolver problemas de privacidade na blockchain.

Além disso, exploramos como o **Zokrates** pode ser usado para criar programas de conhecimento zero para complementar os conceitos teóricos abordados anteriormente neste capítulo.  
Com relação à privacidade, temos muitas soluções, incluindo, mas não se limitando a: **Zcash, Manta, Aleo, ZKOPRU, Aztec e Espresso**.  
Para resolver a escalabilidade, algumas soluções importantes são: **zkSync, Scroll e Polygon Hermez**.

No próximo capítulo, apresentaremos a **segurança**, que é outro desafio que a tecnologia blockchain enfrenta.